Lambda Mu Calculus and Duality: Call-by-Name and Call-by-Value

Jérôme Rocheteau

$\lambda\mu$-calculus and duality: call-by-name and call-by-value

Jérôme Rocheteau

ESTAS – INRETS
20 rue Élisée Reclus – BP 317
F-59666 Villeneuve d'Ascq Cedex
jerome.rocheteau@inrets.fr

Preuves, Programmes et Systèmes
CNRS – Université de Paris VII
UMR 7126 – Case 7014
175 rue du Chevaleret – 75013 Paris – France

Abstract. Under the extension of Curry-Howard's correspondence to classical logic, Gentzen's NK and LK systems can be seen as syntax-directed systems of simple types respectively for Parigot's $\lambda\mu$-calculus and Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus. We aim at showing their computational equivalence. We define translations between these calculi. We prove simulation theorems for an undirected evaluation as well as for call-by-name and call-by-value evaluations.

1 Introduction

Key systems for classical logic in proof theory are Gentzen's NK and LK. The logical equivalence between the latter was proved in [Gentzen, 1934]. We deal with the extension of Curry-Howard's correspondence between proofs and programs through the systems of simple types for the $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-calculi. This extension concerns some other calculi. It is initially Felleisen's λc-calculus. Its type system is the intuitionistic natural deduction with the double negation axiom. Griffin proposed this axiom as the type for the c-operator in [Griffin, 1990]. However, we focus on calculi that correspond closer to Gentzen's systems. The $\lambda\mu$-calculus was defined for NK in [Parigot, 1992]. The $\bar{\lambda}\mu\tilde{\mu}$-calculus was designed for LK in [Curien and Herbelin, 2000]. In the general case, these two calculi are not deterministic. There exists critical pairs. The $\bar{\lambda}\mu\tilde{\mu}$-calculus admits two deterministic projections depending on choosing one of the two possible symmetric orientations of a critical pair. They correspond to the call-by-name/call-by-value duality.

We aim at proving the computational equivalence between $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-calculi. A major step was reached with the proof of the simulation of the $\lambda\mu$-calculus by the $\bar{\lambda}\mu\tilde{\mu}$-calculus in [Curien and Herbelin, 2000]. It holds both for call-by-name and call-by-value evaluations. We present the call-by-name/call-by-value projections of the $\lambda\mu$-calculus in the same way as for the $\bar{\lambda}\mu\tilde{\mu}$ in [Curien and Herbelin, 2000] . It consists of choosing one of the two possible orientations of a critical pair. We prove that the $\lambda\mu$-calculus simulates backwards the $\bar{\lambda}\mu\tilde{\mu}$-calculus in such a way that we obtain easily the same result for the call-by-name, for the call-by-value and for the simple type case. The

$\bar{\lambda}\mu\tilde{\mu}$-calculus is composed of three syntactic categories: terms, contexts (or environments) and commands. The $\lambda\mu$-calculus is basically composed of terms and commands. We add contexts to the $\lambda\mu$-calculus. It eases mappings between the $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-calculi. We extend the translation from the $\lambda\mu$-calculus to the $\bar{\lambda}\mu\tilde{\mu}$-calculus defined in [Curien and Herbelin, 2000] over the $\lambda\mu$-contexts. We define backwards a translation from the $\bar{\lambda}\mu\tilde{\mu}$-calculus to the $\lambda\mu$-calculus.

In section 2 we present the $\lambda\mu$-calculus. In section 3 we present the $\bar{\lambda}\mu\tilde{\mu}$-calculus. In section 4 we define translations between these two calculi. In section 5 we prove simulation theorems that hold for call-by-name and call-by-value.

2 $\lambda\mu$-calculus

We follow the definition given in [Parigot, 1992]. Firstly, we present the grammar of terms and commands. Secondly, we present the system of simple types. Thirdly, we present generic reductions and their call-by-name and call-by-value projections. Fourthly, we extend both the grammar and the type system to the contexts.

Basically, the $\lambda\mu$-calculus is composed of terms and commands. They are defined by mutual induction:

$$t \ ::= \ x \mid \lambda x.t \mid (t)\, t \mid \mu\alpha.c \qquad c \ ::= \ [\alpha]\, t$$

Symbols x range over λ-variables, symbols α range over μ-variables. We note $x \in t$ or $\alpha \in t$ the fact that x or α has a free occurrence in t. Symbols λ and μ are binders. Two terms are equal modulo α-equivalence.

The system of simple types for the $\lambda\mu$-calculus is based on two kinds of sequents. The first $\Gamma \vdash t : T \mid \Delta$ concerns the terms and the second $c : (\Gamma \vdash \Delta)$ concerns the commands in which T is a simple type obtained by the grammar $T \ ::= \ X \mid T \to T$, Γ is a finite domain application from λ-variables to simple types and Δ is a finite domain application from μ-variables to simple types. Γ, Γ' denotes the union of the applications Γ and Γ'. System rules are:

$$\frac{}{x{:}A \vdash x{:}A \mid} \qquad \frac{\Gamma \vdash t{:}B \mid \Delta}{\Gamma \backslash \{x{:}A\} \vdash \lambda x.t{:}A \to B \mid \Delta} \qquad \frac{\Gamma \vdash u{:}A \to B \mid \Delta \qquad \Gamma' \vdash v{:}A \mid \Delta'}{\Gamma, \Gamma' \vdash (u)\, v{:}B \mid \Delta, \Delta'} \ (*)$$

$$\frac{\Gamma \vdash t{:}A \mid \Delta}{[\alpha]\, t{:}(\Gamma \vdash \Delta, \alpha{:}A)} \ (*) \qquad \frac{c{:}(\Gamma \vdash \Delta)}{\Gamma \vdash \mu\alpha.c{:}A \mid \Delta \backslash \{\alpha{:}A\}}$$

The restriction $(*)$ requires that Γ and Γ' match each other on the intersection of their domains. This holds for Δ and Δ' too.

The category of contexts is introduced in order to ease comparisons with the homonymous category of the $\bar{\lambda}\mu\tilde{\mu}$-calculus. $\lambda\mu$-contexts are defined by mutual induction with the terms:

$$e \ ::= \ \alpha \mid \beta(t) \mid t \cdot e$$

We can see contexts as commands with a hole to fill. The first construction α expects a term t in order to provide the command $[\alpha]\, t$. The second $\beta(t)$ expects

a term u in order to provide the command $[\beta]\,(t)\,u$. The last $t \cdot h$ puts the term t on a stack and expects another term to fill the hole.

Definition 1. *Let t a term and e a context. The command $e\{t\}$ is defined by induction on e:*

$$e\{t\} = \begin{cases} [\alpha]\,t & \text{if } e = \alpha \\ [\beta]\,(u)\,t & \text{if } e = \beta(u) \\ h\{(t)\,u\} & \text{if } e = u \cdot h \end{cases}$$

The type system is extended to another kind of sequents $\Gamma \mid e : T \vdash \Delta$. The typing rules give the context e the type of the term t that fills the hole of e:

$$\frac{}{\mid \alpha{:}A \vdash \alpha{:}A} \qquad \frac{\Gamma \vdash t{:}(A \to B) \mid \Delta}{\Gamma \mid \beta(t){:}A \vdash \Delta, \beta{:}B} \qquad \frac{\Gamma \vdash t{:}A \mid \Delta \qquad \Gamma' \mid e{:}B \vdash \Delta'}{\Gamma, \Gamma' \mid t \cdot e{:}(A \to B) \vdash \Delta, \Delta'}$$

A sequent calculus like cut-rule can then be derived in this system as a term against context application.

Lemma 1. *The rule* $\dfrac{\Gamma \vdash t{:}A \mid \Delta \qquad \Gamma' \mid e{:}A \vdash \Delta'}{e\{t\}{:}(\Gamma, \Gamma' \vdash \Delta, \Delta')}$ *holds in $\lambda\mu$.*

Proof. By induction on e.

- if $e = \alpha$ then $e\{t\} = [\alpha]\,t$ and $\dfrac{\Gamma \vdash t{:}A \mid \Delta}{[\alpha]\,t{:}(\Gamma \vdash \Delta, \alpha{:}A)}$
- if $e = \beta(u)$ then $e\{t\} = [\beta]\,(u)\,t$ and

$$\dfrac{\dfrac{\Gamma \vdash u{:}(A \to B) \mid \Delta \qquad \Gamma' \vdash t{:}A \mid \Delta'}{\Gamma, \Gamma' \vdash (u)\,t{:}B \mid \Delta, \Delta'}}{[\beta]\,(u)\,t{:}(\Gamma, \Gamma' \vdash \Delta, \Delta', \beta{:}B)}$$

- if $e = u \cdot h$ then $e\{t\} = h\{(t)\,u\}$ and

$$\dfrac{\dfrac{\Gamma \vdash t{:}(A \to B) \mid \Delta \qquad \Gamma' \vdash u{:}A \mid \Delta'}{\Gamma, \Gamma' \vdash (t)\,u{:}B \mid \Delta, \Delta'} \qquad \Gamma'' \mid h{:}B \vdash \Delta''}{h\{(t)\,u\}{:}(\Gamma, \Gamma', \Gamma'' \vdash \Delta, \Delta', \Delta'')}\ \text{ind. hyp.}$$

Definition 2. *Let t a term, e a context and α a μ-variable, The term $t[\alpha \leftarrow e]$ – the substitution of α by e in t – is defined by induction on t:*

$$t[\alpha \leftarrow e] = \begin{cases} x & \text{if } t = x \\ \lambda x.u[\alpha \leftarrow e] & \text{if } t = \lambda x.u \\ (u[\alpha \leftarrow e])\,v[\alpha \leftarrow e] & \text{if } t = (u)\,v \\ \mu\beta.c[\alpha \leftarrow e] & \text{if } t = \mu\beta.c \end{cases}$$

$$c[\alpha \leftarrow e] = \begin{cases} e\{t[\alpha \leftarrow e]\} & \text{if } c = [\alpha]\,t \\ [\beta]\,t[\alpha \leftarrow e] & \text{if } c = [\beta]\,t \end{cases}$$

The computation notion is based on reductions. We remind one-step reduction rules:

$$\begin{array}{ll} (\lambda x.u)\, t \to_\beta u[x \leftarrow t] & (\mu\alpha.c)\, t \to_\mu \mu\alpha.c[\alpha \leftarrow t \cdot \alpha] \\ \mu\delta.[\delta]\, t \to_\theta t \ (\mathsf{if}\ \delta \notin t) & (t)\, \mu\alpha.c \to_{\mu'} \mu\alpha.c[\alpha \leftarrow \alpha(t)] \\ & [\beta]\, \mu\alpha.c \to_\rho c[\alpha \leftarrow \beta] \end{array}$$

The reduction $\xrightarrow{*}_\gamma$ stands for the reflexive and transitive closure of $\to_\gamma$ and the reduction $\xrightarrow{*}$ stands for the union of $\xrightarrow{*}_\gamma$ for $\gamma \in \{\beta, \mu, \mu', \rho, \theta\}$.

Some of these reductions are linear. Both of the ρ and θ-reductions are linear because they correspond to the identity in NK. The β-reduction from the term $(\lambda x.t)\, y$ is linear because it consists of replacing a variable by another variable inside a term. It corresponds to a normalisation against an axiom rule in NK. The β-reduction from the term $(\lambda x.t)\, u$ where x has a single free occurrence in t is linear too because it consists either of substituting a single variable occurrence by any term. It corresponds either to a normalisation without a proof-tree branch duplication.

Reductions $\rightsquigarrow_\gamma$, $\overset{*}{\rightsquigarrow}_\gamma$ and $\overset{*}{\rightsquigarrow}$ have the same meanings as in the general case. The relation $\approx$ is defined as the reflexive, transitive and symmetric closure of $\overset{*}{\rightsquigarrow}$.

There exists a critical pair for computation determinism. Applicative terms $(\lambda x.t)\, \mu\beta.d$ and $(\mu\alpha.c)\, \mu\beta.d$ can be β or μ'-rewritten in the first case and μ or μ'-rewritten in the second case. We can see the call-by-name and call-by-value disciplines as restrictions of the generic reductions.

The call-by-name evaluation consists of allowing every reduction but the μ'-rule. The β-reduction holds in the first case and the μ-reduction in the second. Formally the call-by-name reduction is $\xrightarrow{*}_n = \xrightarrow{*} \setminus \xrightarrow{*}_{\mu'}$.

The call-by-value evaluation consists of prohibiting β and μ-reductions in which the argument is a μ-abstraction. Formally we define a subset of terms called values by this grammar: $v ::= x \mid \lambda x.t$. β_v and μ_v-reductions are defined instead of generic β and μ ones:

$$(\lambda x.u)\, v \to_{\beta_v} u[x \leftarrow v] \qquad (\mu\alpha.c)\, v \to_{\mu_v} \mu\alpha.c[\alpha \leftarrow v \cdot \alpha]$$

The call-by-value reduction $\xrightarrow{*}_v$ is the union of $\xrightarrow{*}_\gamma$ for $\gamma \in \{\beta_v, \mu_v, \mu', \rho, \theta\}$. Critical pairs are then μ'-rewritten.

There is another way to define call-by-value into the $\lambda\mu$-calculus. The solution is detailed in [Ong and Stewart, 1997]. It consists of restricting the μ'-rule to values instead of the μ:

$$(v)\, \mu\alpha.c \to_{\mu'_v} \mu\alpha.c[\alpha \leftarrow \alpha(v)]$$

Formally $\xrightarrow{*}_v$ becomes the union of $\xrightarrow{*}_\gamma$ for $\gamma \in \{\beta_v, \mu, \mu'_v, \rho, \theta\}$. In fact terms $(\lambda x.t)\, \mu\alpha.c$ and $(\mu\alpha.c)\, \mu\alpha'.c'$ are respectively μ' and μ-reduced because $\mu\alpha.c$ is not a value in these cases. However, we follow Curien-Herbelin's call-by-value definition.

We finish this section by a lemma. It is useful for the section 5 simulation theorems. Any command of the form $e\{\mu\alpha.c\}$ is a redex. However, some can not be reduced in call-by-name nor in call-by-value.

Lemma 2. $e\{\mu\alpha.c\} \xrightarrow{*} c[\alpha \leftarrow e]$

Proof. By induction on e.

- if $e = \beta$ then $e\{\mu\alpha.c\} = [\beta]\,\mu\alpha.c \;\; \leadsto_\rho \;\; c[\alpha \leftarrow \beta]$
- if $e = \beta(t)$ then

$$\begin{array}{rcl} e\{\mu\alpha.c\} & = & [\beta]\,(t)\,\mu\alpha.c \\ & \rightarrow_{\mu'} & [\beta]\,\mu\alpha.c[\alpha \leftarrow \alpha(t)] \\ & \leadsto_\rho & c[\alpha \leftarrow \alpha(t)][\alpha \leftarrow \beta] \\ & = & c[\alpha \leftarrow \beta(t)] \end{array}$$

- if $e = t \cdot h$ then

$$\begin{array}{rcl} e\{\mu\alpha.c\} & = & h\{(\mu\alpha.c)\,t\} \\ & \rightarrow_{\mu} & h\{\mu\alpha.c[\alpha \leftarrow t \cdot \alpha]\} \\ & \xrightarrow{*} & c[\alpha \leftarrow t \cdot \alpha][\alpha \leftarrow h] \\ & = & c[\alpha \leftarrow t \cdot h] \end{array}$$

This lemma does not hold in call-by-name for the $\beta(t)$ induction case because no μ'-rule is allowed. It holds in call-by-value if t is a value for the $h \cdot t$ induction case.

3 $\bar{\lambda}\mu\tilde{\mu}$-calculus

The $\bar{\lambda}\mu\tilde{\mu}$-calculus has the same relation against LK as the $\lambda\mu$-calculus against NK. Reductions of $\bar{\lambda}\mu\tilde{\mu}$-calculus correspond to the cut elimination steps in LK as well as the $\lambda\mu$-reductions correspond to the NK-normalisation. We follow the definition given in [Curien and Herbelin, 2000]. Firstly, we present the grammar of the $\bar{\lambda}\mu\tilde{\mu}$-calculus. Secondly, we present the simple type system. Thirdly, we present generic reductions and their call-by-name and call-by-value projections.

The $\bar{\lambda}\mu\tilde{\mu}$-calculus is basically composed of terms, commands and contexts[1]. They are defined by mutual induction:

$$t \;::=\; x \mid \lambda x.t \mid \mu\alpha.c \qquad c \;::=\; \langle t \,|\, e \rangle \qquad e \;::=\; \alpha \mid t \cdot e \mid \tilde{\mu}x.c$$

As in the $\lambda\mu$, symbols x range over λ-variables, symbols α range over μ-variables and symbols λ, μ and $\tilde{\mu}$ are binders. Terms are equal modulo α-equivalence.

This calculus symmetry looks like LK's left/right symmetry. It is confirmed by its system of simple types. This system shares types with the $\lambda\mu$-calculus. It shares the same kinds of sequents too. Its rules are:

[1] In [Dougherty et al., 2004] these are referred to respectively callers, callees and capsules. We kept the terminology in [Curien and Herbelin, 2000] that sounds closer to its meaning: terms are programs, contexts are environments and commands represent "a closed system containing both the program and its environment".

$$\frac{}{x{:}A\vdash x{:}A \mid} \qquad \frac{}{\mid \alpha{:}A\vdash\alpha{:}A} \qquad \frac{\Gamma\vdash t{:}B \mid \Delta}{\Gamma\backslash\{x{:}A\}\vdash\lambda x.t{:}A\to B \mid \Delta} \qquad \frac{\Gamma\vdash t{:}A \mid \Delta \qquad \Gamma' \mid e{:}B\vdash\Delta'}{\Gamma,\Gamma' \mid t\cdot e{:}A\to B\vdash\Delta,\Delta'}\ (*)$$

$$\frac{c{:}(\Gamma\vdash\Delta)}{\Gamma\vdash\mu\alpha.c{:}A \mid \Delta\backslash\{\alpha{:}A\}} \qquad \frac{c{:}(\Gamma\vdash\Delta)}{\Gamma\backslash\{x{:}A\} \mid \tilde{\mu}x.c{:}A\vdash\Delta} \qquad \frac{\Gamma\vdash t{:}A \mid \Delta \qquad \Gamma' \mid e{:}A\vdash\Delta'}{\langle t \mid e\rangle{:}(\Gamma,\Gamma'\vdash\Delta,\Delta')}\ (*)$$

The restriction $(*)$ is the same as that of $\lambda\mu$.

We present one-step reduction rules. Substitutions inside the $\bar{\lambda}\mu\tilde{\mu}$-calculus are supposed to be known. Each rule concerns a command but the θ-rule:

$$\begin{array}{cc} \langle\lambda x.u \mid t\cdot e\rangle \to_\beta \langle t \mid \tilde{\mu}x.\langle u \mid e\rangle\rangle & \langle\mu\alpha.c \mid e\rangle \to_\mu c[\alpha \leftarrow e] \\ \mu\delta.\langle t \mid \delta\rangle \to_\theta t\ (\delta \notin t) & \langle t \mid \tilde{\mu}x.c\rangle \to_{\tilde{\mu}} c[x \leftarrow t] \end{array}$$

μ and $\tilde{\mu}$-reductions are duals of each other. They correspond to the structural rules in LK. Reductions $\xrightarrow{*}_\gamma$ and $\overset{*}{\rightsquigarrow}_\gamma$ have the same meanings as in the $\lambda\mu$-calculus. The β-rule is a mere term modification without term duplication. It is therefore a linear reduction. The θ-reduction is linear too. There is no ρ-reduction. It is a μ-rule particular case in which $e = \beta$.

This system is not deterministic. There is a single critical pair $\langle\mu\alpha.c \mid \tilde{\mu}x.d\rangle$. It can be both μ or $\tilde{\mu}$-rewritten so that Church-Rosser's property does not hold. In fact $\langle\mu\alpha.\langle x \mid y\cdot\alpha\rangle \mid \tilde{\mu}x.\langle z \mid x\cdot\beta\rangle\rangle$ is μ-rewritten as $\langle x \mid y\cdot\tilde{\mu}x.\langle z \mid x\cdot\beta\rangle\rangle$ and is $\tilde{\mu}$-rewritten as $\langle z \mid \mu\alpha.\langle x \mid y\cdot\alpha\rangle\cdot\beta\rangle$. These are two different normal forms.

Call-by-name and call-by-value disciplines still deal with this problem. They both consist of restricting the context construction. The first new grammar is called $\bar{\lambda}\mu\tilde{\mu}_T$ and the second is called $\bar{\lambda}\mu\tilde{\mu}_Q$.

The call-by-name evaluation consists of restricting the μ-rule to a subset of contexts that are called stacks. $\bar{\lambda}\mu\tilde{\mu}_T$-grammar is:

$$t\ ::=\ x \mid \lambda x.t \mid \mu\alpha.c \quad c\ ::=\ \langle t \mid e\rangle \quad s\ ::=\ \alpha \mid t\cdot s \quad e\ ::=\ s \mid \tilde{\mu}x.c$$

The μ_n-rule is restricted to the stacks:

$$\langle\mu\alpha.c \mid s\rangle \to_{\mu_n} c[\alpha \leftarrow s]$$

Call-by-name reduction $\xrightarrow{*}_n$ is the union of $\xrightarrow{*}_\gamma$ for $\gamma \in \{\beta, \mu_n, \tilde{\mu}, \theta\}$. The critical pair can then only be $\tilde{\mu}$-rewritten. This reduction was proved confluent and stable in the $\bar{\lambda}\mu\tilde{\mu}_T$-calculus in [Curien and Herbelin, 2000].

The call-by-value oriented grammar consists of allowing the $t\cdot e$ context construction only for values. $\bar{\lambda}\mu\tilde{\mu}_Q$-grammar is:

$$t\ ::=\ x \mid \lambda x.t \mid \mu\alpha.c \quad v\ ::=\ x \mid \lambda x.t \quad c\ ::=\ \langle t \mid e\rangle \quad e\ ::=\ \alpha \mid v\cdot e \mid \tilde{\mu}x.c$$

The $\tilde{\mu}_v$-rule is restricted to values:

$$\langle v \mid \tilde{\mu}x.c\rangle \to_{\tilde{\mu}_v} c[x \leftarrow v]$$

Call-by-value reduction $\xrightarrow{*}_v$ is the union of $\xrightarrow{*}_\gamma$ for $\gamma \in \{\beta, \mu, \tilde{\mu}_v, \theta\}$. The command $\langle\mu\alpha.c \mid \mu\alpha'.c'\rangle$ can then only be μ-rewritten. This reduction was proved confluent and stable in the $\bar{\lambda}\mu\tilde{\mu}_Q$-calculus in [Curien and Herbelin, 2000].

The β'-rule contracts as shortcut for both a linear β-rule and a $\tilde{\mu}$-rule:

$$\langle \lambda x.u \,|\, t \cdot e \rangle \to_{\beta'} \langle u[x \leftarrow t] \,|\, e \rangle$$

This β'-rule is obviously compatible with the call-by-name evaluation. It is also compatible with the call-by-value because t is a value by definition of $\bar{\lambda}\mu\tilde{\mu}_Q$.

4 Translations between $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-calculi

We define a translation $(\ )^\dagger$ from $\lambda\mu$ to $\bar{\lambda}\mu\tilde{\mu}$. It extends that of Curien-Herbelin to the $\lambda\mu$-contexts. We define backwards a translation $(\ )^\circ$ from $\bar{\lambda}\mu\tilde{\mu}$ to $\lambda\mu$. We prove properties about their compatibilities with the simple type system and about their compositions.

Definition 3. *Application* $(\ )^\dagger$ *maps any* $\lambda\mu$*-term* t*, command* c *and context* e *respectively to a* $\bar{\lambda}\mu\tilde{\mu}$*-term, command and context.* $(\ )^\dagger$ *is defined by induction on* t*,* c *and* e*:*

$$t^\dagger = \begin{cases} x & \text{if } t = x \\ \lambda x.u^\dagger & \text{if } t = \lambda x.u \\ \mu\beta.\langle v^\dagger \,|\, \tilde{\mu}y.\langle u^\dagger \,|\, y \cdot \beta \rangle \rangle & \text{if } t = (u)\, v \quad (\star) \\ \mu\alpha.c^\dagger & \text{if } t = \mu\alpha.c \end{cases}$$

$$c^\dagger = [\alpha]\, t^\dagger = \langle t^\dagger \,|\, \alpha \rangle$$

$$e^\dagger = \begin{cases} \alpha & \text{if } e = \alpha \\ \tilde{\mu}y.\langle t^\dagger \,|\, y \cdot \beta \rangle & \text{if } e = \beta(t) \quad (\star\star) \\ t^\dagger \cdot h^\dagger & \text{if } e = t \cdot h \end{cases}$$

Condition $(\star)$ requires that variables y and β have no free occurrence in u neither in v. Condition $(\star\star)$ requires that $y \notin t$. A straightforward induction leads us to state that t and $t^\dagger$ have the same free variables set.

It seems more natural to translate $(u)\, v$ by $\mu\beta.\langle u^\dagger \,|\, v^\dagger \cdot \beta \rangle$. This shorter term corresponds in LK to the arrow elimination rule in NK too. But it would not be compatible with the call-by-value evaluation. For example, $(x)\, \mu\alpha.c$ would be translated as $\mu\beta.\langle x \,|\, \mu\alpha.c^\dagger \cdot \beta \rangle$ in this case. It can not be reduced by any rule in the $\bar{\lambda}\mu\tilde{\mu}$-calculus. However, $(x)\, \mu\alpha.c$ can be μ'-reduced in the $\lambda\mu$-calculus.

$(u)\, v$ should be translated as $\mu\beta.\langle u^\dagger \,|\, \tilde{\mu}y.\langle v^\dagger \,|\, \tilde{\mu}x.\langle y \,|\, x \cdot \beta \rangle \rangle \rangle$ with Ong and Stewart's call-by-value definition in [Ong and Stewart, 1997].

We show that translation $(\ )^\dagger$ is compatible with the type system. If a typing environment for a term t exists, it holds for $t^\dagger$.

Lemma 3. $\Gamma \vdash t : A \mid \Delta \implies \Gamma \vdash t^\dagger : A \mid \Delta$

Proof. By a straightforward induction on t. We show the less than obvious cases.

– if $t = (u)\, v$ then $t^\dagger = \mu\beta.\langle v^\dagger \mid \tilde{\mu}y.\langle u^\dagger \mid y \cdot \beta\rangle\rangle$ and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y{:}A \vdash y{:}A \mid}\quad\dfrac{}{\mid \beta{:}B \vdash \beta{:}B}}{y{:}A \mid y{\cdot}\beta{:}(A{\to}B) \vdash \beta{:}B}}{\Gamma \vdash u^\dagger{:}A{\to}B \mid \Delta \qquad \langle u^\dagger \mid y{\cdot}\beta\rangle{:}(\Gamma, y{:}A \vdash \Delta, \beta{:}B)}}{\Gamma \mid \tilde{\mu}y.\langle u^\dagger \mid y{\cdot}\beta\rangle \vdash \Delta, \beta{:}B}}{\Gamma' \vdash v^\dagger{:}A \mid \Delta' \qquad \langle v^\dagger \mid \tilde{\mu}y.\langle u^\dagger \mid y{\cdot}\beta\rangle\rangle{:}(\Gamma, \Gamma' \vdash \Delta, \Delta', \beta{:}B)}}{\Gamma, \Gamma' \vdash \mu\beta.\langle v^\dagger \mid \tilde{\mu}y.\langle u^\dagger \mid y{\cdot}\beta\rangle\rangle{:}B \mid \Delta, \Delta'}$$

– if $e = \beta(t)$ then $e^\dagger = \tilde{\mu}y.\langle t^\dagger \mid y \cdot \beta\rangle$ and

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y{:}A \vdash y{:}A \mid}\quad\dfrac{}{\mid \beta{:}B \vdash \beta{:}B}}{y{:}A \mid y{\cdot}\beta{:}(A{\to}B) \vdash \beta{:}B}}{\Gamma \vdash t^\dagger{:}(A{\to}B) \mid \Delta \qquad \langle t^\dagger \mid y{\cdot}\beta\rangle{:}(\Gamma, y{:}A \vdash \Delta, \beta{:}B)}}{\Gamma \mid \tilde{\mu}y.\langle t^\dagger \mid y{\cdot}\beta\rangle{:}A \vdash \Delta, \beta{:}B}$$

Definition 4. *Application* $(\ )^\circ$ *maps backwards any* $\bar{\lambda}\mu\tilde{\mu}$*-term* t *to a* $\lambda\mu$*-term. Definition 1 is used to translate any* $\bar{\lambda}\mu\tilde{\mu}$*-command* c*. Definition of the* $\lambda\mu$*-contexts is used to map the* $\bar{\lambda}\mu\tilde{\mu}$*-contexts* e *as well.* $(\ )^\circ$ *is built by induction on* t*,* c *and* e*:*

$$t^\circ = \begin{cases} x & \text{if } t = x \\ \lambda x.u^\circ & \text{if } t = \lambda x.u \\ \mu\alpha.c^\circ & \text{if } t = \mu\alpha.c \end{cases}$$

$$c^\circ = \langle t \mid e\rangle^\circ = e^\circ\{t^\circ\}$$

$$e^\circ = \begin{cases} \alpha & \text{if } e = \alpha \\ t^\circ \cdot h^\circ & \text{if } e = t \cdot h \\ \beta(\lambda x.\mu\delta.c^\circ) & \text{if } e = \tilde{\mu}x.c \quad (*) \end{cases}$$

Condition $(*)$ requires that $\delta \notin c$. t and t° have the same free variables set. Application $(\ )^\circ$ is compatible with the type system too.

Lemma 4. $\Gamma \vdash t : A \mid \Delta \implies \Gamma \vdash t^\circ : A \mid \Delta$

Proof. By a straightforward induction on t. We give two cases.

– if $c = \langle t \mid e\rangle$ then $c^\circ = e^\circ\{t^\circ\}$ and

$$\dfrac{\Gamma \vdash t{:}A \mid \Delta \qquad \Gamma' \mid e{:}A \vdash \Delta'}{\langle t \mid e\rangle{:}(\Gamma, \Gamma' \vdash \Delta, \Delta')}^\circ = \dfrac{\Gamma \vdash t^\circ{:}A \mid \Delta \qquad \Gamma' \mid e^\circ{:}A \vdash \Delta'}{e^\circ\{t^\circ\}{:}(\Gamma, \Gamma' \vdash \Delta, \Delta')}\ \text{lem. 1}$$

– if $e = \tilde{\mu}x.c$ then $e^\circ = \beta(\lambda x.\mu\delta.c^\circ)$ and

$$\dfrac{c{:}(\Gamma \vdash \Delta)}{\Gamma\backslash\{x{:}A\} \mid \tilde{\mu}x.c{:}A \vdash \Delta}^\circ = \dfrac{\dfrac{\dfrac{c^\circ{:}(\Gamma \vdash \Delta)}{\Gamma \vdash \mu\delta.c^\circ{:}B \mid \Delta}}{\Gamma\backslash\{x{:}A\} \vdash \lambda x.\mu\delta.c^\circ{:}(A{\to}B)}}{\Gamma\backslash\{x{:}A\} \mid \beta(\lambda x.\mu\delta.c^\circ){:}A \vdash \Delta, \beta{:}B}$$

We focus on properties about the composition of $(\ )^\dagger$ and $(\ )^\circ$. We want to state that $t^{\dagger\circ} = t$ and that $t^{\circ\dagger} = t$ for any term. But it is not the case, these results hold modulo linear reductions.

Theorem 1. $t^{\dagger\circ} \overset{*}{\rightsquigarrow} t$

Proof. By a straightforward induction on t. Every cases is obtained successively by expanding definitions 3, 1, 4 and by applying the induction hypothesis. We give the case which uses linear reductions additionally.

– if $t = (u)\, v$ then

$$\begin{array}{rcl}
(u)\, v^{\dagger\circ} & = & \mu\beta.\langle v^\dagger \mid \tilde{\mu}y.\langle u^\dagger \mid y \cdot \beta\rangle\rangle^\circ \\
& = & \mu\beta.[\gamma]\, (\lambda y.\mu\delta.[\beta]\, (u^{\dagger\circ})\, y)\, v^{\dagger\circ} \\
& \overset{*}{\rightsquigarrow} & \mu\beta.[\gamma]\, (\lambda y.\mu\delta.[\beta]\, (u)\, y)\, v \\
& \rightsquigarrow_\beta & \mu\beta.[\gamma]\, \mu\delta.[\beta]\, (u)\, v \\
& \rightsquigarrow_\rho & \mu\beta.[\beta]\, (u)\, v \\
& \rightsquigarrow_\theta & (u)\, v
\end{array}$$

We prove two lemmas before stating backwards that $(\ )^{\circ\dagger}$ is the identity modulo linear reductions. The first lemma is useful to prove the second.

Lemma 5. $\langle t_0 t_1 \dots t_n{}^\dagger \mid e\rangle \overset{*}{\rightsquigarrow} \langle t_0{}^\dagger \mid t_1{}^\dagger \cdot \dots \cdot t_n{}^\dagger \cdot e\rangle$

Proof. By induction on n.

– if $n = 0$ then it is obvious
– if $n = m + 1$ then

$$\begin{array}{rcl}
\langle t_0 t_1 \dots t_m t_{m+1}{}^\dagger \mid e\rangle & = & \langle \mu\beta.\langle t_{m+1}{}^\dagger \mid \tilde{\mu}y.\langle t_0 t_1 \dots t_m{}^\dagger \mid y \cdot \beta\rangle\rangle \mid e\rangle \\
& \rightsquigarrow_\mu & \langle t_{m+1}{}^\dagger \mid \tilde{\mu}y.\langle t_0 t_1 \dots t_m{}^\dagger \mid y \cdot e\rangle\rangle \\
& \rightsquigarrow_{\tilde{\mu}} & \langle t_0 t_1 \dots t_m{}^\dagger \mid t_{m+1}{}^\dagger \cdot e\rangle \\
& \overset{*}{\rightsquigarrow} & \langle t_0{}^\dagger \mid t_1{}^\dagger \cdot \dots \cdot t_m{}^\dagger \cdot t_{m+1}{}^\dagger \cdot e\rangle
\end{array}$$

The second lemma shows how to map a definition 1 command.

Lemma 6. $e\{t\}^\dagger \overset{*}{\rightsquigarrow} \langle t^\dagger \mid e^\dagger\rangle$

Proof. By induction on e.

– if $e = \alpha$ then it is obvious by definitions 1 and 3
– if $e = \beta(u)$ then

$$\begin{array}{rcl}
\beta(u)\{t\}^\dagger & = & [\beta]\, (u)\, t^\dagger \\
& = & \langle \mu\gamma.\langle t^\dagger \mid \tilde{\mu}y.\langle u^\dagger \mid y \cdot \gamma\rangle\rangle \mid \beta\rangle \\
& \rightsquigarrow_\mu & \langle t^\dagger \mid \tilde{\mu}y.\langle u^\dagger \mid y \cdot \beta\rangle\rangle \\
& = & \langle t^\dagger \mid \beta(u)^\dagger\rangle
\end{array}$$

– if $e = u \cdot h$ then

$$\begin{array}{rcl} u \cdot h\{t\}^\dagger & = & h\{(t)\, u\}^\dagger \\ & \overset{*}{\rightsquigarrow} & \langle (t)\, u^\dagger \mid h^\dagger \rangle \\ & \overset{*}{\rightsquigarrow} & \langle t^\dagger \mid u^\dagger \cdot h^\dagger \rangle \\ & = & \langle t^\dagger \mid u \cdot h^\dagger \rangle \end{array}$$

Theorem 2. $t^{\circ\dagger} \overset{*}{\rightsquigarrow} t$

Proof. By induction on t. We apply definitions 3, 4 successively and the induction hypothesis. We give a typical case and another which needs either the previous lemma or linear reductions.

– if $c = \langle t \mid e \rangle$ then

$$\begin{array}{rcl} \langle t \mid e \rangle^{\circ\dagger} & = & e^\circ\{t^\circ\}^\dagger \\ & \overset{*}{\rightsquigarrow} & \langle t^{\circ\dagger} \mid e^{\circ\dagger} \rangle \\ & \overset{*}{\rightsquigarrow} & \langle t \mid e \rangle \end{array}$$

– if $e = \tilde{\mu}x.c$ then

$$\begin{array}{rcl} \tilde{\mu}x.c^{\circ\dagger} & = & \beta(\lambda x.\mu\beta.c^\circ)^\dagger \\ & = & \tilde{\mu}y.\langle \lambda x.\mu\beta.c^{\circ\dagger} \mid y \cdot \beta \rangle \\ & \overset{*}{\rightsquigarrow} & \tilde{\mu}y.\langle \lambda x.\mu\beta.c \mid y \cdot \beta \rangle \\ & \rightsquigarrow_\beta & \tilde{\mu}y.\langle y \mid \tilde{\mu}x.\langle \mu\beta.c \mid \beta \rangle \rangle \\ & \rightsquigarrow_{\tilde{\mu}} & \tilde{\mu}x.\langle \mu\beta.c \mid \beta \rangle \\ & \rightsquigarrow_\mu & \tilde{\mu}x.c \end{array}$$

5 Simulations between $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-calculi

We want to prove that the $\lambda\mu$-calculus simulates and is simulated backwards by the $\bar{\lambda}\mu\tilde{\mu}$-calculus. We focus on the undirected evaluation. Call-by-name and call-by-value are drawn from this.

We begin with the simulation of the $\lambda\mu$ by the $\bar{\lambda}\mu\tilde{\mu}$. The next four lemmas show results of a $\lambda\mu$-substitution after a β, μ, μ' and ρ-reduction. Each proof consists successively of

– expanding the $\lambda\mu$-substitution
– expanding the definition of $(\ )^\dagger$
– applying the induction hypothesis if necessary
– factorising the $\bar{\lambda}\mu\tilde{\mu}$-substitution
– factorising the definition of $(\ )^\dagger$

We give basic cases and those which use lemmas additionally for any proof.

Lemma 7. $t[x \leftarrow u]^\dagger = t^\dagger[x \leftarrow u^\dagger]$

Proof. By induction on t.

– if $t = x$ then $x[x \leftarrow u]^\dagger = u^\dagger = x^\dagger[x \leftarrow u^\dagger]$

– if $t = y$ then $y[x \leftarrow u]^\dagger = y = y^\dagger[x \leftarrow u^\dagger]$
– if $t = (v)\, w$ then

$$\begin{aligned}(v)\, w[x \leftarrow u]^\dagger &= (v[x \leftarrow u])\, w[x \leftarrow u]^\dagger \\ &= \mu\beta.\langle w[x \leftarrow u]^\dagger \mid \tilde{\mu}y.\langle v[x \leftarrow u]^\dagger \mid y \cdot \beta\rangle\rangle \\ &= \mu\beta.\langle w^\dagger[x \leftarrow u^\dagger] \mid \tilde{\mu}y.\langle v^\dagger[x \leftarrow u^\dagger] \mid y \cdot \beta\rangle\rangle \\ &= \mu\beta.\langle w^\dagger \mid \tilde{\mu}y.\langle v^\dagger \mid y \cdot \beta\rangle\rangle[x \leftarrow u^\dagger] \\ &= (v)\, w^\dagger[x \leftarrow u^\dagger]\end{aligned}$$

Lemma 8. $t[\alpha \leftarrow u \cdot \alpha]^\dagger \overset{*}{\rightsquigarrow} t^\dagger[\alpha \leftarrow u^\dagger \cdot \alpha]$

Proof. By induction on t.

– if $t = (a)\, b$ then

$$\begin{aligned}(a)\, b[\alpha \leftarrow u \cdot \alpha]^\dagger &= (a[\alpha \leftarrow u \cdot \alpha])\, b[\alpha \leftarrow u \cdot \alpha]^\dagger \\ &= \mu\beta.\langle b[\alpha \leftarrow u \cdot \alpha]^\dagger \mid \tilde{\mu}y.\langle a[\alpha \leftarrow u \cdot \alpha]^\dagger \mid y \cdot \beta\rangle\rangle \\ &\overset{*}{\rightsquigarrow} \mu\beta.\langle b^\dagger[\alpha \leftarrow u^\dagger \cdot \alpha] \mid \tilde{\mu}y.\langle a^\dagger[\alpha \leftarrow u^\dagger \cdot \alpha] \mid y \cdot \beta\rangle\rangle \\ &= \mu\beta.\langle b^\dagger \mid \tilde{\mu}y.\langle a^\dagger \mid y \cdot \beta\rangle\rangle[\alpha \leftarrow u^\dagger \cdot \alpha] \\ &= (a)\, b^\dagger[\alpha \leftarrow u^\dagger \cdot \alpha]\end{aligned}$$

– if $c = [\alpha]\, w$ then

$$\begin{aligned}[\alpha]\, w[\alpha \leftarrow u \cdot \alpha]^\dagger &= [\alpha]\, w[\alpha \leftarrow u \cdot \alpha]^\dagger \\ &= u \cdot \alpha\{w[\alpha \leftarrow u \cdot \alpha]\}^\dagger \\ &= \langle w[\alpha \leftarrow u \cdot \alpha]^\dagger \mid u^\dagger \cdot \alpha\rangle \\ &\overset{*}{\rightsquigarrow} \langle w^\dagger[\alpha \leftarrow u^\dagger \cdot \alpha] \mid u^\dagger \cdot \alpha\rangle \\ &= \langle w^\dagger \mid \alpha\rangle[\alpha \leftarrow u^\dagger \cdot \alpha] \\ &= [\alpha]\, w^\dagger[\alpha \leftarrow u^\dagger \cdot \alpha]\end{aligned}$$

Lemma 9. $t[\alpha \leftarrow \alpha(u)]^\dagger \overset{*}{\rightsquigarrow} t^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle y \mid y \cdot \alpha\rangle]u^\dagger$

Proof. By induction on t.

– if $t = (a)\, b$ then

$$\begin{aligned}(a)\, b[\alpha \leftarrow \alpha(u)]^\dagger &= (a[\alpha \leftarrow \alpha(u)])\, b[\alpha \leftarrow \alpha(u)]^\dagger \\ &= \mu\beta.\langle b[\alpha \leftarrow \alpha(u)]^\dagger \mid \tilde{\mu}y.\langle a[\alpha \leftarrow \alpha(u)]^\dagger \mid y \cdot \beta\rangle\rangle \\ &\overset{*}{\rightsquigarrow} \mu\beta.\langle b^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle] \mid \tilde{\mu}y.\langle a^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle] \mid y \cdot \beta\rangle\rangle \\ &= \mu\beta.\langle b^\dagger \mid \tilde{\mu}y.\langle a^\dagger \mid y \cdot \beta\rangle\rangle[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle] \\ &= (a)\, b^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle]\end{aligned}$$

– if $c = [\alpha]\, w$ then

$$\begin{aligned}[\alpha]\, w[\alpha \leftarrow \alpha(u)]^\dagger &= [\alpha]\, (w[\alpha \leftarrow \alpha(u)])\, u^\dagger \\ &= \langle (w[\alpha \leftarrow \alpha(u)])\, u^\dagger \mid \alpha\rangle \\ &\overset{*}{\rightsquigarrow} \langle w[\alpha \leftarrow \alpha(u)]^\dagger \mid u^\dagger \cdot \alpha\rangle \\ &\overset{*}{\rightsquigarrow} \langle w^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle] \mid u^\dagger \cdot \alpha\rangle \\ &= \langle w^\dagger \mid \alpha\rangle[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle] \\ &= [\alpha]\, w^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle u^\dagger \mid y \cdot \alpha\rangle]\end{aligned}$$

Lemma 10. $t[\alpha \leftarrow \beta]^\dagger = t^\dagger[\alpha \leftarrow \beta]$

Proof. By induction on t.

– if $c = [\alpha]\, u$ then

$$\begin{aligned}
[\alpha]\, u[\alpha \leftarrow \beta]^\dagger &= [\beta]\, u[\alpha \leftarrow \beta]^\dagger \\
&= \langle u[\alpha \leftarrow \beta]^\dagger \mid \beta \rangle \\
&= \langle u^\dagger[\alpha \leftarrow \beta] \mid \beta \rangle \\
&= \langle u^\dagger \mid \alpha \rangle [\alpha \leftarrow \beta] \\
&= [\alpha]\, u^\dagger[\alpha \leftarrow \beta]
\end{aligned}$$

Theorem 3 (simulation of the $\lambda\mu$-calculus by the $\bar{\lambda}\mu\tilde{\mu}$-calculus).

$$t \to_\gamma v \implies \exists u\ t^\dagger \xrightarrow{*} u \ \wedge\ v^\dagger \overset{*}{\rightsquigarrow} u$$

Proof. By cases on γ.

– if $\gamma = \beta$ then

$$\begin{array}{rcl}
(\lambda x.u)\, v^\dagger & = & \mu\beta.\langle v^\dagger \mid \tilde{\mu}y.\langle \lambda x.u^\dagger \mid y \cdot \beta \rangle \rangle \\
& \rightsquigarrow_\beta & \mu\beta.\langle v^\dagger \mid \tilde{\mu}y.\langle y \mid \tilde{\mu}x.\langle u^\dagger \mid \beta \rangle \rangle \rangle \\
& \rightsquigarrow_{\tilde{\mu}} & \mu\beta.\langle v^\dagger \mid \tilde{\mu}x.\langle u^\dagger \mid \beta \rangle \rangle \\
& \to_{\tilde{\mu}} & \mu\beta.\langle u^\dagger[x \leftarrow v^\dagger] \mid \beta \rangle \\
& \rightsquigarrow_\theta & u^\dagger[x \leftarrow v^\dagger] \\
& = & u[x \leftarrow v]^\dagger
\end{array}$$

– if $\gamma = \mu$ then

$$\begin{array}{rcl}
(\mu\alpha.c)\, v^\dagger & = & \mu\beta.\langle v^\dagger \mid \tilde{\mu}y.\langle \mu\alpha.c^\dagger \mid y \cdot \beta \rangle \rangle \\
& \rightsquigarrow_{\tilde{\mu}} & \mu\alpha.\langle \mu\alpha.c^\dagger \mid v^\dagger \cdot \alpha \rangle \\
& \to_\mu & \mu\alpha.c^\dagger[\alpha \leftarrow v^\dagger \cdot \alpha] \\
& \approx & \mu\alpha.c[\alpha \leftarrow v \cdot \alpha]^\dagger
\end{array}$$

– if $\gamma = \mu'$ then

$$\begin{array}{rcl}
(v)\, \mu\alpha.c^\dagger & = & \mu\beta.\langle \mu\alpha.c^\dagger \mid \tilde{\mu}y.\langle v^\dagger \mid y \cdot \beta \rangle \rangle \\
& \to_\mu & \mu\alpha.c^\dagger[\alpha \leftarrow \tilde{\mu}y.\langle v^\dagger \mid y \cdot \alpha \rangle] \\
& \approx & \mu\alpha.c[\alpha \leftarrow \alpha(v)]^\dagger
\end{array}$$

– if $\gamma = \rho$ then

$$\begin{array}{rcl}
[\beta]\, \mu\alpha.c^\dagger & = & \langle \mu\alpha.c^\dagger \mid \beta \rangle \\
& \rightsquigarrow_\mu & c^\dagger[\alpha \leftarrow \beta] \\
& = & c[\alpha \leftarrow \beta]^\dagger
\end{array}$$

– if $\gamma = \theta$ then $\mu\delta.[\delta]\, t^\dagger = \mu\delta.\langle t^\dagger \mid \delta \rangle \ \rightsquigarrow_\theta\ t^\dagger$

Corollary 1 (call-by-name case). $t \to_n v \implies \exists u\ t^\dagger \xrightarrow{*}_n u \ \wedge\ v^\dagger \overset{*}{\rightsquigarrow}_n u$

Proof. By cases on β and μ-rules.

$(\lambda x.u)\, v$ is β-reduced in call-by-name without any restriction. It is simulated in the $\bar{\lambda}\mu\tilde{\mu}$-calculus by a $\tilde{\mu}$-reduction. The latter is in call-by-name without any restriction too.

$(\mu\alpha.c)\, v$ is μ-reduced in call-by-name without any restriction. It is simulated in the $\bar{\lambda}\mu\tilde{\mu}$-calculus by a μ-reduction. The latter is in call-by-name if $v^\dagger \cdot \alpha$ is a stack. It is the case by definition 3.

Corollary 2 (call-by-value case). $t \to_v v \implies \exists u\ t^\dagger \xrightarrow{*}_v u \ \wedge\ v^\dagger \overset{*}{\leadsto}_v u$

Proof. By cases on β, μ and μ'-rules.

$(\lambda x.u)\, v$ is β-reduced in call-by-value if v is a value. It is simulated in the $\bar{\lambda}\mu\tilde{\mu}$-calculus by a $\tilde{\mu}$-reduction. The latter is in call-by-value if $v^\dagger$ is a value. It is the case by the definition of $\bar{\lambda}\mu\tilde{\mu}_Q$.

$(\mu\alpha.c)\, v$ is μ-reduced in call-by-value if v is a value. It is simulated in the $\bar{\lambda}\mu\tilde{\mu}$-calculus by a μ-reduction. The latter is in call-by-value without any restriction.

$(v)\, \mu\alpha.c$ is μ'-reduced in call-by-value without any restriction. It is simulated in the $\bar{\lambda}\mu\tilde{\mu}$-calculus by a μ-reduction. The latter is in call-by-value without any restriction as well.

The $\bar{\lambda}\mu\tilde{\mu}$-simulation by the $\lambda\mu$-calculus requires preliminary lemmas showing that $(\ )^\circ$ commutes over $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-substitutions. Each proof consists of

- expanding the $\bar{\lambda}\mu\tilde{\mu}$-substitution
- expanding the definition of $(\ )^\circ$
- applying the induction hypothesis if necessary
- factorising the $\lambda\mu$-substitution
- factorising the definition of $(\ )^\circ$

Lemma 11. $t[x \leftarrow u]^\circ = t^\circ[x \leftarrow u^\circ]$

Proof. By induction on t.

- if $t = x$ then $x[x \leftarrow u]^\circ = u^\circ = x^\circ[x \leftarrow u^\circ]$
- if $t = y$ then $y[x \leftarrow u]^\circ = y = y^\circ[x \leftarrow u^\circ]$
- if $t = \langle t \,|\, e \rangle$ then

$$\begin{aligned}
\langle t \,|\, e\rangle[x \leftarrow u]^\circ &= \langle t[x \leftarrow u] \,|\, e[x \leftarrow u]\rangle^\circ \\
&= e[x \leftarrow u]^\circ\{t[x \leftarrow u]^\circ\} \\
&= e^\circ[x \leftarrow u^\circ]\{t^\circ[x \leftarrow u^\circ]\} \\
&= e^\circ\{t^\circ\}[x \leftarrow u^\circ] \\
&= \langle t \,|\, e\rangle^\circ[x \leftarrow u^\circ]
\end{aligned}$$

Lemma 12. $t[\alpha \leftarrow h]^\circ = t^\circ[\alpha \leftarrow h^\circ]$

Proof. By induction on t.

– if $c = \langle t \,|\, e \rangle$ then

$$\begin{aligned}\langle t \,|\, e\rangle[\alpha \leftarrow h]^\circ &= \langle t[\alpha \leftarrow h] \,|\, e[\alpha \leftarrow h]\rangle^\circ \\ &= e[\alpha \leftarrow h]^\circ\{t[\alpha \leftarrow h]^\circ\} \\ &= e^\circ[\alpha \leftarrow h^\circ]\{t^\circ[\alpha \leftarrow h^\circ]\} \\ &= e^\circ\{t^\circ\}[\alpha \leftarrow h^\circ] \\ &= \langle t \,|\, e\rangle^\circ[\alpha \leftarrow h^\circ]\end{aligned}$$

– if $e = \alpha$ then $\alpha[\alpha \leftarrow h]^\circ = h^\circ = \alpha^\circ[\alpha \leftarrow h^\circ]$
– if $e = \beta$ then $\beta[\alpha \leftarrow h]^\circ = \beta^\circ = \beta^\circ[\alpha \leftarrow h^\circ]$

Theorem 4 (simulation of the $\bar{\lambda}\mu\tilde{\mu}$-calculus by the $\lambda\mu$-calculus).

$$t \to_\gamma v \Longrightarrow \exists u \; t^\circ \xrightarrow{*} u \; \overset{*}{\rightsquigarrow} \; v^\circ$$

Proof. By cases on γ.

– if $\gamma = \beta'$ then

$$\begin{aligned}\langle \lambda x.u \,|\, v \cdot e\rangle^\circ &= v^\circ \cdot e^\circ\{\lambda x.u^\circ\} \\ &= e^\circ\{(\lambda x.u^\circ)\, v^\circ\} \\ &\to_\beta e^\circ\{u^\circ[x \leftarrow v^\circ]\} \\ &= e^\circ\{u[x \leftarrow v]^\circ\} \\ &= \langle u[x \leftarrow v] \,|\, e\rangle^\circ\end{aligned}$$

– if $\gamma = \mu$ then

$$\begin{aligned}\langle \mu\alpha.c \,|\, e\rangle^\circ &= e^\circ\{\mu\alpha.c^\circ\} \\ &\xrightarrow{*} c^\circ[\alpha \leftarrow e^\circ] \\ &= c[\alpha \leftarrow e]^\circ\end{aligned}$$

– if $\gamma = \tilde{\mu}$ then

$$\begin{aligned}\langle t \,|\, \tilde{\mu}x.c\rangle^\circ &= [\beta]\,(\lambda x.\mu\delta.c^\delta)\, t^\circ \\ &\to_\beta [\beta]\,\mu\delta.c^\circ[x \leftarrow t^\circ] \\ &\rightsquigarrow_\rho c^\circ[x \leftarrow t^\circ] \\ &= c[x \leftarrow t]^\circ\end{aligned}$$

– if $\gamma = \theta$ then $\mu\delta.\langle t \,|\, \delta\rangle^\circ = \mu\delta.[\delta]\, t^\circ \;\; \rightsquigarrow_\theta \;\; t^\circ$

Corollary 3 (call-by-name case). $t \to_n v \Longrightarrow \exists u \; t^\circ \xrightarrow{*}_n u \; \overset{*}{\rightsquigarrow}_n \; v^\circ$

Proof. By cases on β', μ and $\tilde{\mu}$-rules.

$\langle \lambda x.u \,|\, v \cdot e\rangle$ is β'-reduced in call-by-name without any restriction. It is simulated in the $\lambda\mu$-calculus by a β-reduction. The latter is in call-by-name without any restriction too.

$\langle \mu\alpha.c \,|\, e\rangle$ is μ-reduced in call-by-name if $e \neq \tilde{\mu}x.c'$ else it were $\tilde{\mu}$-reduced. It is simulated in the $\lambda\mu$-calculus with the help of lemma 2. The latter is in call-by-name if $e^\circ \neq \beta(t)$ i.e. if $e \neq \tilde{\mu}x.c'$. It is the case by definition 4.

$\langle t \,|\, \tilde{\mu}x.c\rangle$ is $\tilde{\mu}$-reduced in call-by-name without any restriction. It is simulated in the $\lambda\mu$-calculus by a β-reduction. The latter is in call-by-name without any restriction as well.

Corollary 4 (call-by-value case). $t \to_v v \Longrightarrow \exists u\ t^\circ \overset{*}{\to}_v u\ \overset{*}{\rightsquigarrow}_v\ v^\circ$

Proof. By cases on β', μ and $\tilde{\mu}$-rules.

$\langle \lambda x.u \,|\, v \cdot e \rangle$ is β'-reduced in call-by-value if v is a value. It is simulated in the $\lambda\mu$-calculus by a β-reduction. The latter is in call-by-value if v° is a value. It is the case by definition 4.

$\langle \mu\alpha.c \,|\, e \rangle$ is μ-reduced in call-by-value if e is either a μ-variable or a context of the form $v \cdot h$ where v is a value or a μ-abstraction by the definition of $\bar{\lambda}\mu\tilde{\mu}_Q$. It is simulated in the $\lambda\mu$-calculus with the help of lemma 2. The latter is in call-by-value if v° is a value in a context of the form $h^\circ \cdot v^\circ$ i.e. if v is a value in a $v \cdot h$ context. It is the case by definition 4.

$\langle t \,|\, \tilde{\mu}x.c \rangle$ is $\tilde{\mu}$-reduced in call-by-value if t is a value. It is simulated in the $\lambda\mu$-calculus by a β-reduction. The latter is in call-by-value if t° is a value. It is the case by definition 4.

6 Conclusion

Analysis of the $\lambda\mu$ and $\bar{\lambda}\mu\tilde{\mu}$-calculi has shown their computational equivalence. It holds for undirected evaluations of pure calculi (see theorems 3 and 4). This result is then easily obtained for call-by-name and call-by-value evaluations (see corollaries 1, 2, 3 and 4). It concerns the simple type system too (see lemmas 3 and 4).

The simulation of the $\bar{\lambda}\mu\tilde{\mu}$-calculus by the $\lambda\mu$-calculus is smoother than the simulation of the $\lambda\mu$-calculus by the $\bar{\lambda}\mu\tilde{\mu}$-calculus. The first is obtained with the help of linear *reductions* whereas the second is obtained with the help of linear *expansions*.

This work can be extended in three ways. The first consists of proving the same results for the call-by-value evaluation of the $\lambda\mu$-calculus defined in [Ong and Stewart, 1997]. The second consists of defining CPS translations to λ-calculus in order to complete [Curien and Herbelin, 2000]. The third consists of extending the type system to the other logical constants.

References

Curien and Herbelin, 2000. Curien, P.-L. and Herbelin, H. (2000). The Duality of Computation. In *Proceedings of the International Conference on Functional Programming*.

Dougherty et al., 2004. Dougherty, D. J., Ghilezan, S., and Lescanne, P. (2004). Characterizing strong normalization in a language with control operators. In *Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 155–166.

Gentzen, 1934. Gentzen, G. (1934). Investigations into Logical Deduction. In Szabo, M., editor, *Collected Papers of Gerhard Gentzen*. North Holland.

Griffin, 1990. Griffin, T. G. (1990). The Formulae-as-Types Notion of Control. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages*, pages 47–57. ACM Press.

Ong and Stewart, 1997. Ong, L. and Stewart, C. (1997). A Curry-Howard foundation for functional computation with control. In *Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages*, pages 215–227. ACM Press.

Parigot, 1992. Parigot, M. (1992). $\lambda\mu$-calculus: an Algorithmic Interpretation of Classical Natural Deduction. In *Proceedings of Internationnal Conference on Logic Programming and Automated Deduction*, volume 624 of *Lectures Notes in Computer Science*, pages 190–201. Springer.